INSA DE LYON – DEPARTEMENT INFORMATIQUE

# Projet SCOR

## Documentation technique & retour d'expérience

**BALAIS Olivier & DEFLAUX Julien & ESCANDELL Stéphane & THERME Jérôme**

**Février 2010**

# SOMMAIRE

# PRESENTATION DU PROJET

## A. Rappel du contexte

Le projet se place dans une phase amont à un éventuel nouveau TP pour les futures promotions. Le département informatique nous met à disposition un ensemble de matériel technique (ordinateurs, support de travail, environnement, robots, ...) dans le but de faire jouer des robots Khepera à un jeu semblable au volley.

Pour cela, deux équipes (concurrentes) de quatre personnes ont été formées. Pour chaque équipe, l'objectif est de programmer les robots pour que, lors du match prévu en fin de projet, ceux-ci soient autonomes et remportent la partie (selon les règles du jeu établies). Mr Beslon a pour cela préparé un plateau de jeu destiné à recevoir les robots et les caméras.

Techniquement, chaque équipe a à sa disposition le matériel suivant :

- Un ordinateur sous environnement Linux pour piloter les robots
- Deux robots Khepera reliés par le port USB au PC
- Une webcam reliée par le port USB au PC pour analyser le terrain de jeu

Ce projet fait appel à des notions de :

- Analyse d'image
- Intelligence artificielle
- Communication sur port Série
- Modélisation
- Gestion de projet / équipe

Outre l'aspect purement technique de développement, ce projet spécifique contient une importante partie "gestion de projet" car les étudiants sont laissés libres sur le plan des choix techniques et organisationnels. En effet, étant donné le caractère "expérimental" du projet, il arrive que les équipes doivent faire face à des problèmes imprévus (positionnement de la caméra non optimal, robots aux capacités limités, terrain non fonctionnel, ...). Pour résoudre ces problèmes, un accord entre les deux équipes est dégagé et soumis à l'approbation de Mr Beslon. En ce sens, les élèves sont laissés très libres et sont une force de proposition (un ensemble d'outils est même laissé à leur disposition pour pouvoir, si besoin est, faire des travaux sur le terrain de jeu).

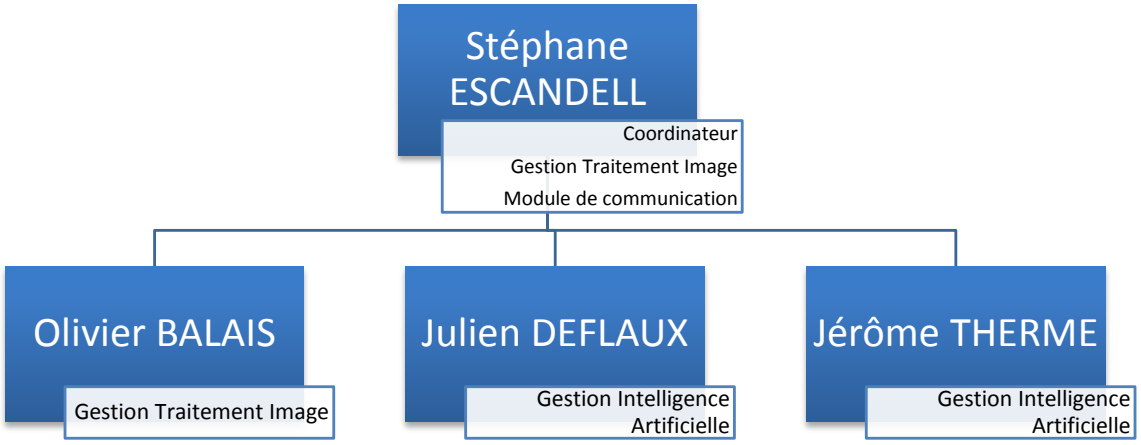
L'objectif final du projet est de prototyper une application de contrôle des robots, d'étudier la faisabilité d'une compétition mais aussi et surtout de mieux cerner les contraintes du projet, d'identifier les difficultés et de proposer des solutions (règles de jeu à modifier, structure du terrain à changer, ...). A l'issue de ce projet, les enseignants devraient disposer de suffisamment de connaissance pour intégrer pleinement cette "coupe de robotique" dans la formation IF et, pourquoi pas, préparer ainsi les étudiants à participer à une réelle coupe de robotique internationale.

Le présent rapport est une documentation sur le travail réalisé par l'une des deux équipes ayant participé au projet pour l'année scolaire 2009/2010 (présentée ci-après). Il est organisé en deux grandes parties :

- Une partie technique présentant l'architecture de l'application mise en place ainsi que les choix techniques effectués. Ce document est donc un support supplémentaire aux fichiers sources normalement fournis ;
- Une partie de retour sur expérience plus destinée aux enseignants. Cette partie précise quels sont les points où nous avons pu buter et quels sont les différents éléments à modifier pour que le projet puisse être viable.

## B. Présentation équipe

Très succinctement, ce dossier est le rapport de projet de l'équipe suivante :




Pour toute information complémentaire, un contact peut être établi avec ces membres via leur adresse INSA respective (prénom.nom@insa-lyon.fr). Si le contexte le leur permet, ils devraient pouvoir être aptes à répondre à toute question.

# DOCUMENTATION TECHNIQUE

## A. Outils de travail

Dans cette partie du dossier, nous présentons les outils informatiques utilisés pour notre développement. Comme nous allons le montrer, les outils et bibliothèques utilisés sont des classiques du genre pour ce type d'applications.

### 1. Système d'exploitation

Lorsque nous avons effectué notre développement nous étions sous environnement Linux. La distribution utilisée était une Ubuntu 9.10. Le compilateur utilisé était celui fourni par défaut, le plus à jour possible selon le gestionnaire de paquet intégré à l'OS.

### 2. Eclipse

Pour gérer notre projet, pouvoir développer à plusieurs en parallèle et aisément débugger le programme, nous avons utilisé l'environnement de développement Eclipse avec le plugin de gestion de projet en C++. Pour ce logiciel nous n'avons pas utilisé la version des paquets de Ubuntu, mais bien celui directement téléchargeable sur le site du constructeur.

### 3. OpenCV 2.0

Comme nous l'avons précisé plus haut, le projet intègre toute une partie de récupération d'images via une webcam et de traitement de ces images. Naturellement, c'est la bibliothèque OpenCV qui s'avérait la plus intéressante. Dans notre équipe, nous avons utilisé la version 2.0 de OpenCV. Pour l'installation de celle-ci, le site officiel est assez bien fait pour que nous ne détaillons pas la démarche sous linux un classique **./configure,make,sudo make install** fonctionne relativement bien). Cela dit, nous allons préciser la méthode d'intégration de OpenCV à Eclipse.

Une fois un projet C++ créé sous Eclipse, ouvrir les propriétés du projet puis :

1- Dans C/C++ Build : sélectionner Settings
2- Dans GCC C++ Compiler dans Directories ajouter **/usr/local/include/opencv**
3- Dans GCC C++ Linker :
    a. Dans librairies ajouter :
        i. Library Search Path : **/usr/local/lib**
        ii. Libraries : cv, cvaux, cxcore, highgui
    b. Dans Miscellanous ajouter **lcxcore**, **lcv**, **lhighgui**

Pour se familiariser avec cette bibliothèque, nous conseillons également le tutoriel du site du zéro spécialement dédié à OpenCV. Dans notre équipe nous n'avons pas pris le temps de le lire, mais il s'avère en réalité un excellent moyen de prise en main de l'outil (compter une dizaine d'heures pour le lire).

## B. Architecture générale

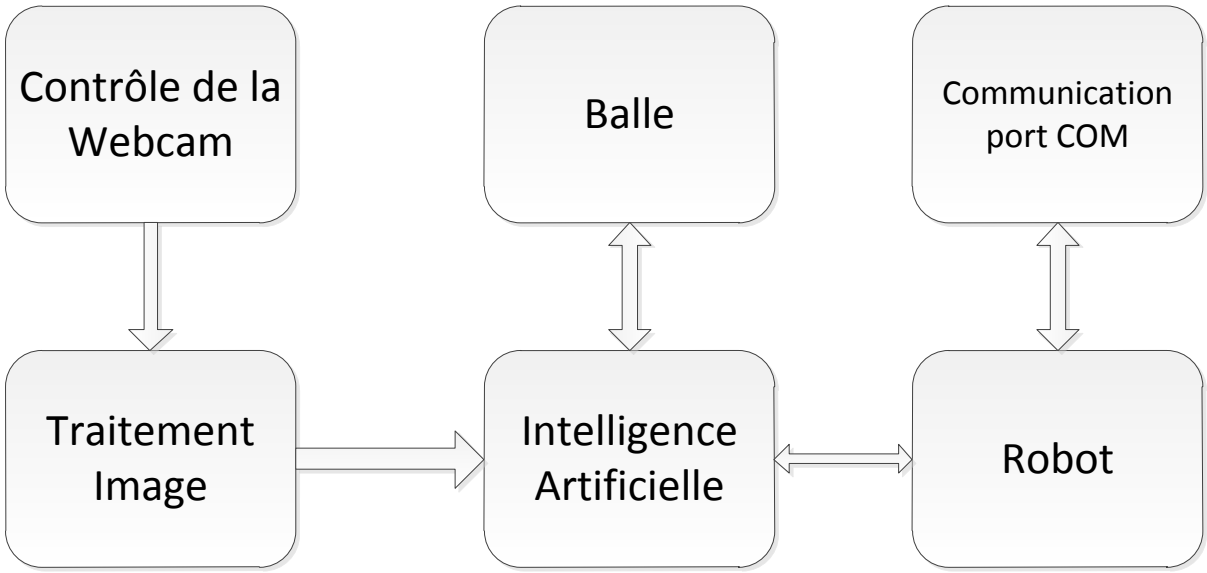
Dans ce projet nous avons identifié plusieurs éléments clés :

- Des éléments mobiles (robots et balle)
- Un protocole de communication entre les robots et les PCs

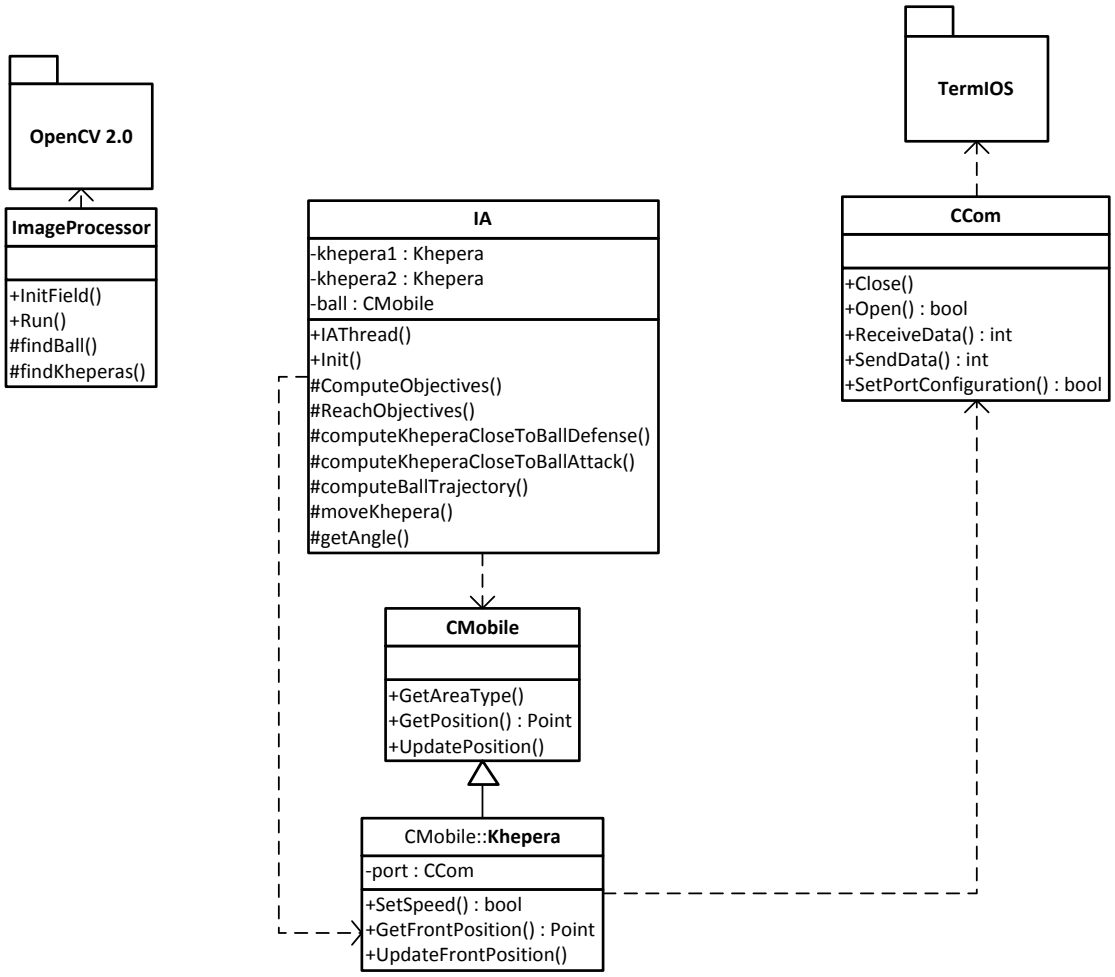
- Un système d'acquisition d'image
- Un système de traitement d'image
- Une intelligence artificielle

Chacun de ces éléments correspond à un module bien distinct. Dans les fichiers fournis, chaque élément correspond à une classe plus ou moins générique.

Schématiquement, les interactions entre ces modules sont les suivantes :




On remarque ainsi immédiatement que le système d'intelligence artificielle est au centre du programme. Entre les choix de bibliothèques et les choix techniques personnels, nous avons finalement mis en place les classes suivantes :

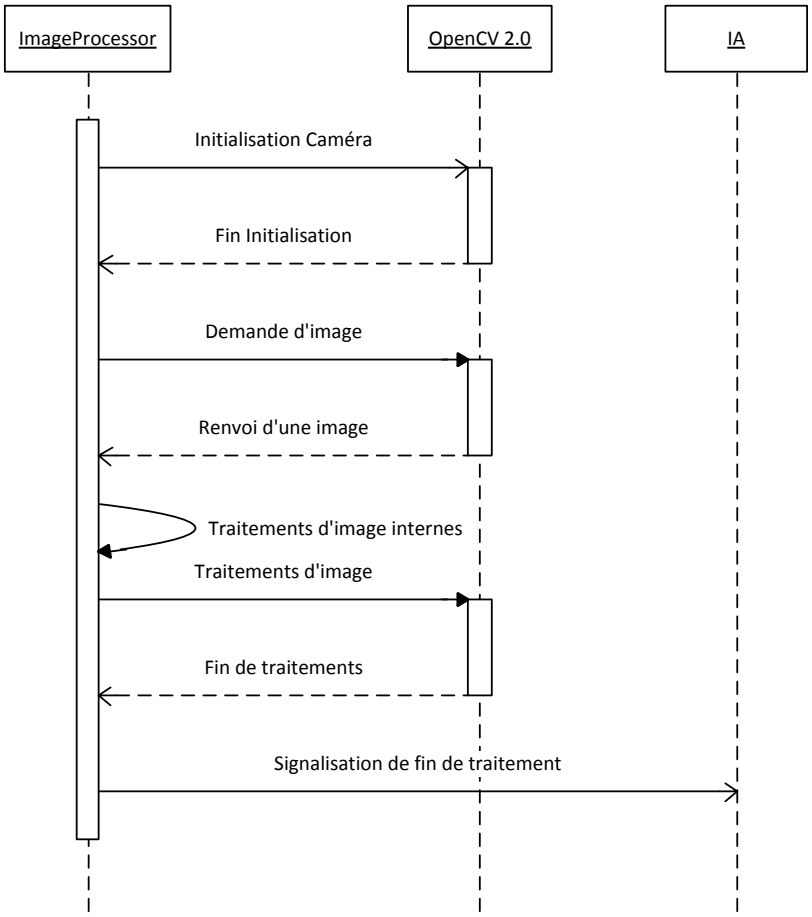


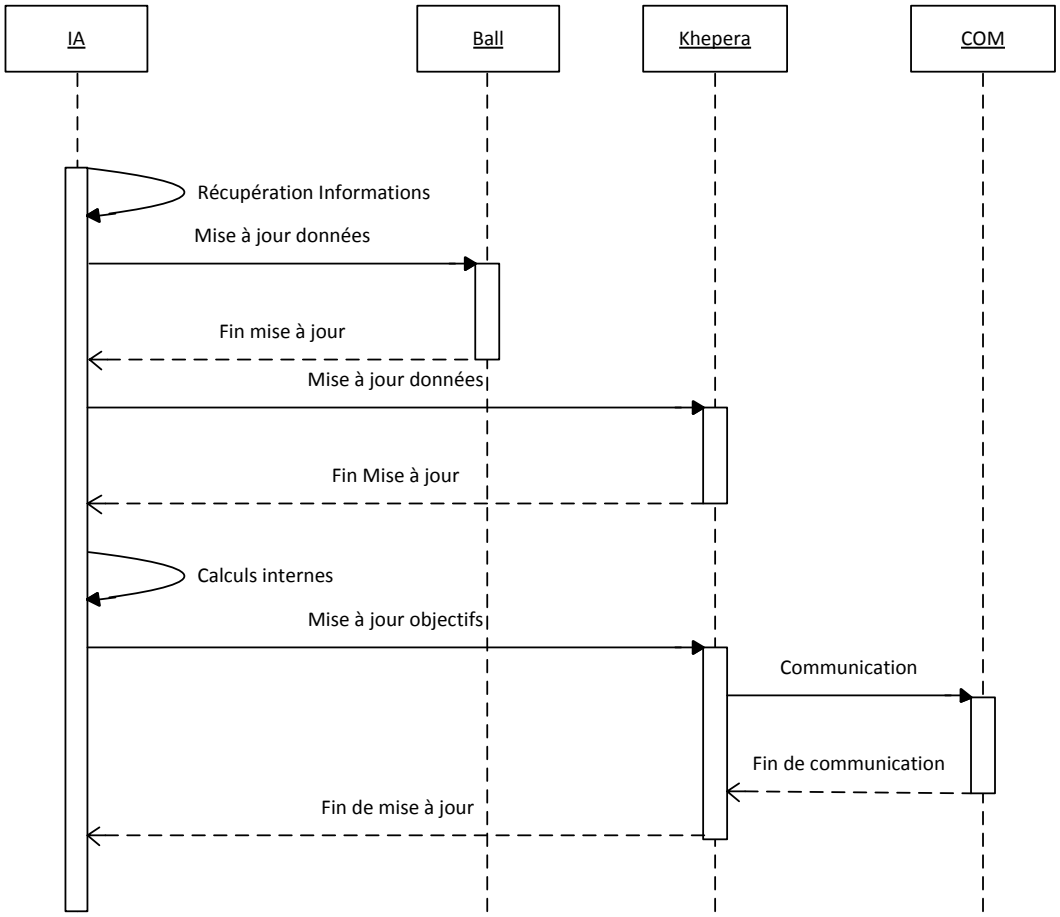

Le diagramme ci-dessus est incomplet. Nous n'avons fait apparaître que les éléments essentiels au projet. Notamment, à cet ensemble, nous avons une classe template Point.tpp (pour manipuler assez facilement un objet point) et un fichier de programme principal RealScor.cpp pour ordonnancer tout l'ensemble.

La classe ImageProcessor est entièrement statique. IA et ImageProcessor sont exécutés dans deux processus différents :

- ImageProcessor dans un processus fils du main ;
- IA est le processus principal ;

Pour avoir une vision globale du fonctionnement du logiciel, les différentes séquences sont les suivantes :


ImageProcessor
OpenCV 2.0
IA
Initialisation Caméra
Fin Initialisation
Demande d'image
Renvoi d'une image
Traitements d'image internes
Traitements d'image
Fin de traitements
Signalisation de fin de traitement




Les diagrammes sont très schématiques, mais ils devraient permettre de bien saisir le principe de fonctionnement de notre application. Toutes ces actions se répètent ensuite en boucle à l'infini.

La communication entre **ImageProcessor** et **IA** se fait via une boîte aux lettres initialisée dans le processus principal (plus de détails sur son contenu ci-après). A la fin de chaque traitement d'une image, **ImageProcessor** dépose un message dans cette boîte. **IA** reste bloqué en lecture sur celle-ci jusqu'à ce qu'un message soit disponible.

## C. Module de communication

Dans cette partie du document, nous nous intéressons de plus prêt au module de communication entre les Kheperas et le PC.

Pour le projet, nous utilisons en réalité un port USB sur lequel est branché un adaptateur USB <-> port COM. Sous Ubuntu, lorsque ces adaptateurs sont branchés, on a alors accès à l'interface COM (virtuelle donc) via le fichier **/dev/ttyUSBX** où X vaut 0 pour le premier branché, 1 pour le second et ainsi de suite.

L'avantage d'être sous un système Linux est que tout est fichier. La communication avec le port COM en est donc facilitée. Dans le cadre de ce projet, nous avons créé une classe complète permettant de gérer un port COM spécialement pour les Kheperas. La classe est presque générique pour pouvoir être utilisée avec n'importe quel autre appareil. Un tout petit effort de modification peut permettre

de rompre le lien existant entre les Kheperas et la classe **CCOM** (seule la méthode de réception des donnés n'est pas générique).

La classe a été réalisée en s'appuyant sur la bibliothèque de Linux spécialement dédiée à cet effet **TermIOS**. Pour plus d'informations sur cette classe, il faudra donc s'en référer à la documentation de TermIOS.

Actuellement à la création d'un objet de type Khepera, celui-ci crée automatiquement un objet CCOM lui permettant de communiquer avec le port COM. Dans l'état actuel, seule la méthode de définition de la vitesse des moteurs est définie. Celle-ci envoie la commande sur le port COM (au vrai Khepera) et attend en retour la réponse de celui-ci (**attention :** peut être source de problème … à modifier en ajoutant une protection par timeout).

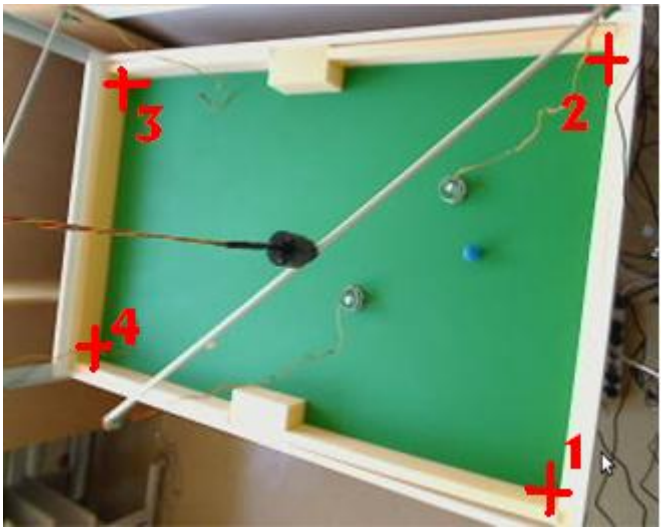
# D. Traitement des images

## 1. Principe général

Au lancement du programme, une fenêtre représentant l'image vue par la webcam est affichée (méthode ImageProcessor: :InitField() appelée depuis le main). Celle-ci permet de définir au programme les quatre coins du terrain pour ensuite :

1- Pouvoir faire une transformation de celui-ci (depuis un parallélogramme vers un vrai rectangle) ;
2- Pouvoir travailler sur l'image, uniquement dans la zone qui nous intéresse ;

Pour cela, il faut cliquer dans la fenêtre aux quatre coins du terrain en commençant par le coin inférieur droit puis en tournant dans le sens antihoraire :




Ceci fait, le programme se lance en mode de jeu. Nous détaillons ici uniquement l'ensemble des traitements effectuées sur les images.

Tout ce qui est affichage d'image, dessin de cercles, récupération des images depuis la webcam, transformation du terrain est effectué grâce à la bibliothèque OpenCV. ImageProcessor possède une méthode ImageProcessor::Run qui boucle indéfiniment et dont les actions s'enchaînent ainsi :

1- Récupération d'une image depuis la webcam via la méthode **cvQueryFrame()** ;

2- Application d'une transformation à l'image (pour travailler dans un rectangle) via les méthodes **cvGetPerspectiveTransform()** et **cvWarpPerspective()** ;
3- Recherche de la balle via un calcul de barycentre (calcul manuel) ;
4- Recherche des Kheperas via une méthode de clustering (calcul manuel) ;
5- Ajout d'un message dans la boîte aux lettres de communication avec l'IA
6- Fin de boucle

## 2. Recherche de la balle

Dans notre équipe, nous n'avons malheureusement pas exploité le potentiel de OpenCV pour pouvoir retrouver des objets sur certains critères (couleur, forme, mouvement, …). L'apprentissage de cette bibliothèque étant quelque peu long, nous avons opté pour faire notre propre moyen de recherche. Voici le processus de recherche de la balle :

1- Sur l'image issue de la webcam (après transformation), nous ne gardons que les pixels à majorité de la couleur de la balle (pour nous le rouge). Nous obtenons alors une nouvelle image monochrome ;
2- On effectue une érosion (via la méthode **cvErode()** de OpenCV) afin d'éliminer le bruit ;
3- Sur cette image, nous calculons le barycentre des points blancs. Le point obtenu correspond au centre de la balle ;

Nous travaillons là avec le système de codage des couleurs RVB.

## 3. Recherche des Kheperas

Pour rechercher les Kheperas le principe est le même :

1- On filtre une couleur ;
2- On recherche les points de concentration

Sauf que pour les Kheperas on a deux nouvelles contraintes :

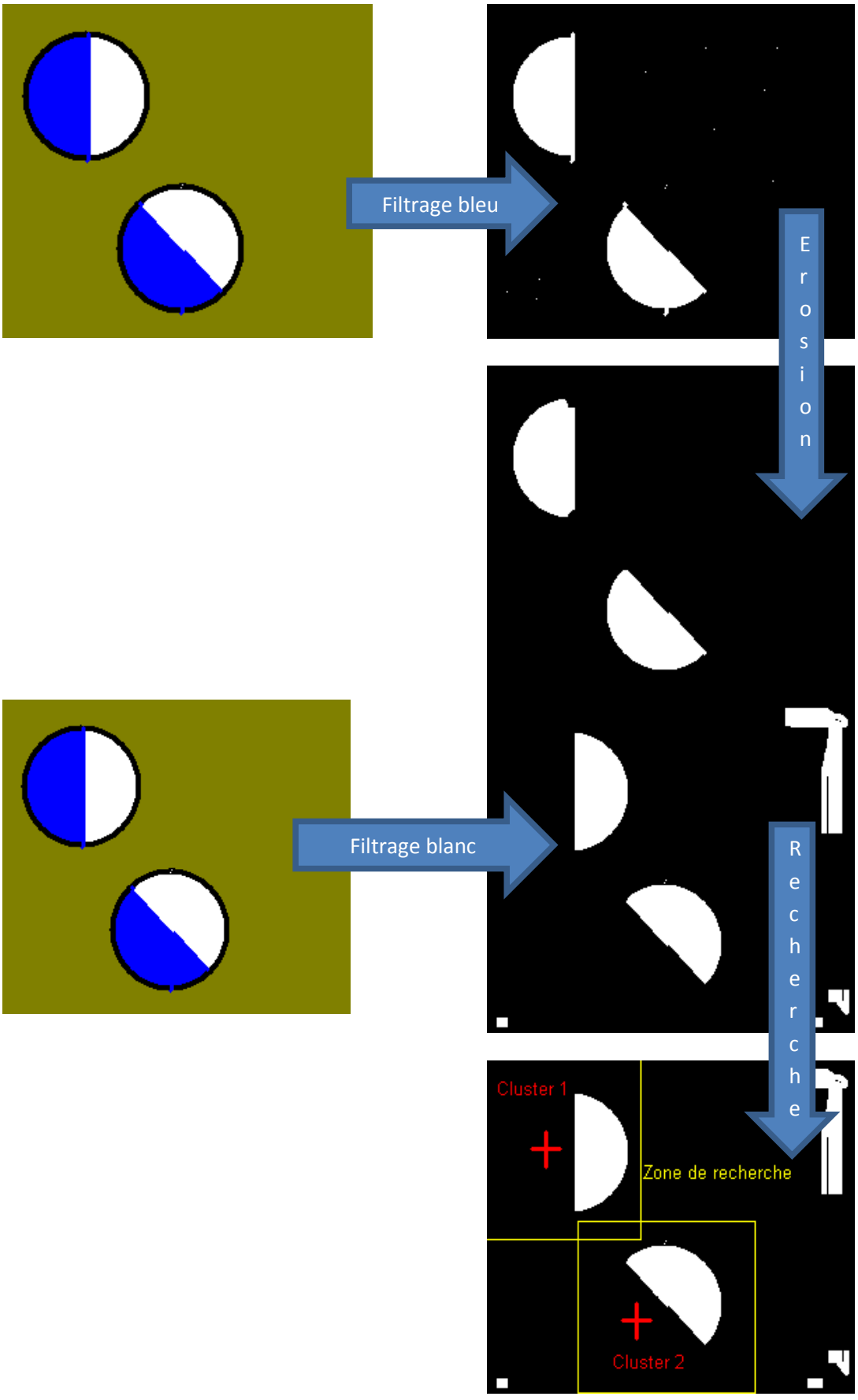
1- On recherche deux objets de la même couleur sur le terrain ;
2- On doit être capable d'identifier l'avant de l'arrière du Khepera ;

Pour cela, chaque Khepera avait sur son dessus deux demi-cercles : un bleu et un blanc pour identifier l'avant. Ainsi la recherche des Kheperas s'effectuait comme suit :

1- Sur l'image issue de la webcam (après transformation), nous ne gardons que les pixels à majorité de bleu. Nous obtenons alors une nouvelle image monochrome ;
2- On effectue une érosion afin d'éliminer le bruit ;
3- On recherche ensuite deux clusters (nous ne prenions pas en compte les kheperas de l'adversaire) selon un calcul de clustering classique ;
4- Sur l'image issue de la webcam (après transformation), nous ne gardons que les pixels à majorité de blanc. Nous obtenons alors une nouvelle image monochrome ;
5- Dans un carrée suffisamment grand et centré sur chaque cluster identifié précédemment, on recherche une concentration de points blancs : cela nous donne alors l'avant du robot ;

Une fois toutes ces informations identifiées, un message est transmis à l'IA

Schématiquement on a le traitement suivant :


Filtrage bleu
Erosion
Filtrage blanc
Recherche
Cluster 1
Zone de recherche
Cluster 2

Lors du calcul de clustering, la toute première fois, les deux points initiaux sont pris au hasard (plus ou moins). Par la suite, les points initiaux sont les derniers points identifiés.

Là encore tous les traitements sont effectués sur le codage de couleur RVB.

### 4. Messages dans la boîte aux lettres

Afin de pouvoir faire communiquer le module de traitement d'image avec le module d'intelligence artificielle, nous avons mis en place une boîte aux lettres pour pouvoir faire passer les informations identifiées par le module de traitement d'image.

Concrètement, les messages dans la boîte aux lettres ont la structure suivante :

```
struct t info
{
    int x ball; // Coordonnée en X de la balle
    int y ball; // Coordonnée en Y de la balle

    int k1 f x; // Coordonnée en X de l'avant du Khepera 1
    int k1 f y; // Coordonnée en Y de l'avant du Khepera 1
    int k1 x;   // Coordonnée en X de l'arrière du Khepera 1
    int k1 y;   // Coordonnée en Y de l'arrière du Khepera 1

    int k2 f x; // cf. ci-dessus pour Khepera 2
    int k2 f y;
    int k2 x;
    int k2 y;
};
```

## E. Intelligence artificielle

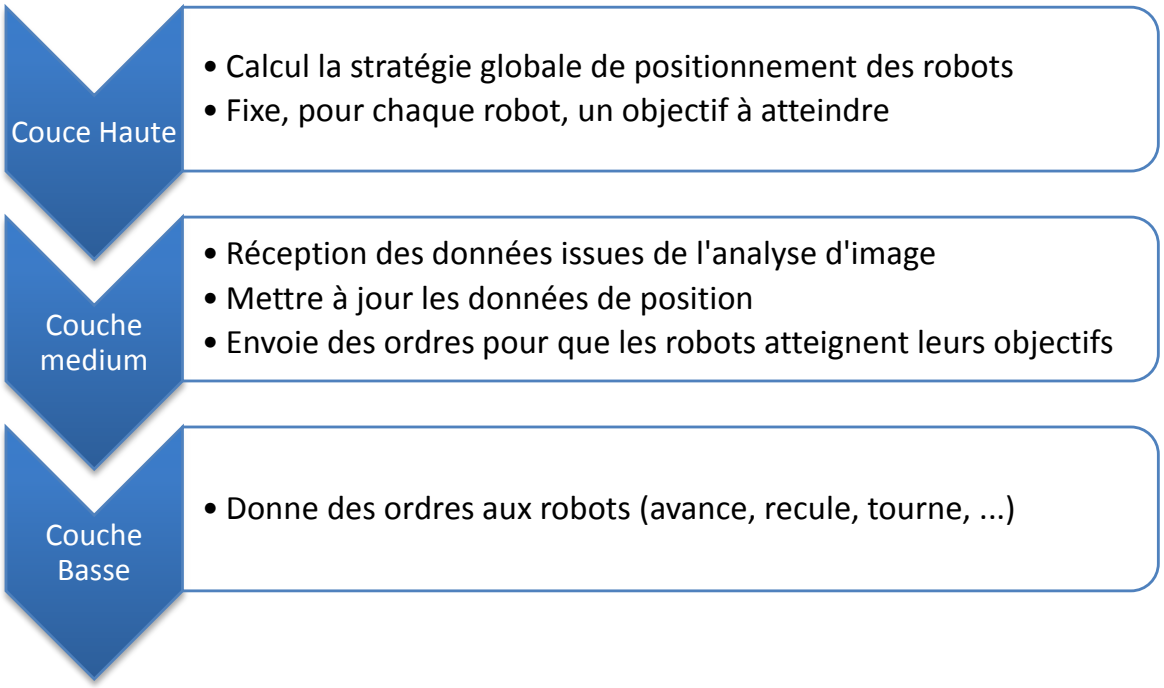
### 1. Architecture idéale en couches

La programmation de robots pour effectuer un match (ou tournois de foot) étant une activité déjà effectuée et pratiquée lors de tournois mondiaux (MIROSOT…) opposants différentes universités à travers le monde, nous nous sommes dans un premier temps inspirés de leur compte rendu pour élaborer notre architecture.

Il en ressort que majoritairement il y a deux grands choix : opter pour une intelligence scriptée et totalement définie par les développeurs, ou alors mettre en place un réseau de neurones qui va apprendre à se déplacer et s'organiser seul.

Dans la plupart des tournois effectués à l'heure actuelle, les gagnants avaient en général optés pour une intelligence scriptée et donc déterministe. Dans le cadre de ce projet nous avons donc suivis leur exemple notamment à cause des contraintes de temps.

Il en ressort donc que l'architecture de l'Intelligence Artificielle est décomposée en trois couches :




## 2. Architecture utilisée

### Fonctionnement global

Pour notre architecture, le fonctionnement en couches décrit ci-dessus a été suivi dans la philosophie, mais pas en pratique pour ne pas complexifier le code inutilement. En effet, il est bon de rappeler que l'objectif ayant été réduit à une séance de tirs aux buts, beaucoup de contraintes imposées par ces couches auraient étés inutiles.

Il y a donc un module seul d'IA, un module de gestion des images et un module qui fournit l'API pour faire avancer les robots.

Pour rappel, le module qui traite l'image reçue par la webcam émet un message pour signaler qu'il a fini ses traitements et que l'IA peut agir. Ces messages sont transmis à une boite aux lettres qui sert d'interface entre l'IA et les données visuelles traitées (positions des robots, orientations des robots, balles).

Le module d'IA est lancé dans un thread qui boucle en continue entre l'attente d'un message à traiter et le traitement de ce message. Le traitement de ce message, détaillé dans la partie suivante, consiste en le calcul des objectifs à atteindre pour chacun de nos kheperas et à les faire avancer vers ces objectifs.

### Algorithme de l'IA expliqué

#### Calcul des objectifs

Le calcul d'un objectif à atteindre pour chaque khepera permet de séparer le calcul des positions et le fait de faire bouger un khepera distinct.

Ce calcul effectue les opérations suivantes :

- Les positions (balle, kheperas, avant des kheperas pour l'orientation) contenus dans le message à traiter, envoyé par le module d'image, sont sauvegardées pour pouvoir calculer les trajectoires plus tard. Dans notre cas nous avons seulement utilisé la trajectoire de la balle, même si le stockage des positions des kheperas est effectué. Nous avons fait le choix de ne stocker que deux positions pour pouvoir calculer simplement la trajectoire de la balle entre deux instants, car les images reçues et donc les messages arrivent suffisamment rapidement.

- Le sens de déplacement est alors calculé pour savoir si la balle est fixe, si elle vient dans notre camp ou si elle s'en éloigne. Nous nous basons sur un seuil fixé de manière expérimentale à partir duquel une balle est définie comme en mouvement.

- Ensuite, différentes stratégies d'attaque ou de défense sont envisagées. Il faudra donc définir les objectifs à atteindre des kheperas qui sont soit leur position initiale soit un point calculé. Dans notre cas, nous nous sommes limités à faire aller les kheperas vers la balle. Les stratégies de kheperas attaquant et défenseurs ont été prévues, mais par manque de temps il a été impossible de fixer des seuils (ou de revoir le fonctionnement) ce qui entrainé des kheperas trop perfectionnistes (comprendre qui n'attaquaient pas réellement la balle)…

### Atteindre les objectifs

Une fois le calcul des objectifs à atteindre effectué, il faut faire déplacer les robots vers ces points. Pour cela, il faut :

- Calculer l'angle orienté que fait le khepera par rapport à la balle. Un calcul simple (mais complexe à formuler) est détaillé dans le code permet d'arriver à calculer cet angle.

- En fonction de cette angle les robots vont soit avancer vers leur objectif, soit tourner dans le bon sens. Lorsque les robots se mettent à tourner, nous les lançons dans un sens sans préciser de combien il faut tourner. En effet, les calculs s'effectuant suffisamment souvent, lorsque les robots auront atteints la bonne position ils avanceront. En pratique ils ne font pas de tours complets, il suffit de régler les seuils de tolérance et de rapidité des roues.

## F. Evolutions du projet

Dans cette partie du dossier, nous précisons, en vrac, les évolutions du projet fourni pour que celui-ci soit plus performant :

- Asservissement robots : actuellement, les robots ont un comportement un peu bête et méchant : on leur donne une vitesse de déplacement et ils exécutent. Une manière évoluée de penser les choses serait de créer un réel asservissement des robots basé sur un objectif à atteindre, leur trajectoire actuelle, leur vitesse … Egalement, il pourrait peut-être être intéressant de contrôler les robots via des ordres différents. Dans l'état actuel, on définit une vitesse de roue. On pourrait imaginer un contrôle de robot par nombre de tour de roues à effectuer ;
- Passage en mode HSV plutôt que RVB : dans le traitement d'image, tout est effectué dans le domaine RGB. Ce domaine est très rapidement limité (influence de la luminosité trop

importante, seuil trop précis, …). Un moyen significatif d'améliorer le traitement d'image serait d'effectuer des traitements dans le domaine HSV ;

- Généricité port COM : fait très facultatif, une méthode du port COM pourrait être modifiée afin de rendre cette classe totalement générique ;
- Utiliser les méthodes **FindContours()** de OpenCV plutôt que des traitements personnels tels que le clustering ou le calcul de barycentre. En effet, les méthodes de OpenCV s'avèreront très certainement plus précises et plus puissantes ;
- Transférer la méthode **GetAngle()** dans la classe template **Point.tpp** : d'un point de vue purement logique de programmation ;
- Créer un moyen de quitter proprement l'application : actuellement l'interception de la touche ESC ne fonctionne pas. Ce qui d'une certaine façon sous-entend que le programme ne se quitte pas proprement (boite aux lettres toujours existante …) ;
- Configurer la caméra : avec OpenCV 2 il est possible de contrôler réellement la caméra et ses paramètres. Il pourrait être intéressant d'approfondir l'intérêt de la méthode **cvCalibrateCamera2()** pour récupérer une image directement transformée en rectangle ;

# RETOUR D'EXPERIENCE

## A. Les retours positifs

### 1. L'intérêt

Le projet SCOR fait appel à 2 principales composantes. D'un côté on trouve le traitement de l'image et de l'autre l'intelligence artificielle. Chacun de ces sujets sont abordés durant la formation du département informatique mais l'application à des cas concrets n'est pas approfondie. L'intérêt est donc évident, le projet permet à l'étudiant de découvrir plus en détail ces technologies.

De plus ce sont des sujets très actuels sur lesquels de nombreux travaux sont réalisés et dont les résultats sont visibles directement dans la vie de tous les jours. Cela ajoute à la motivation l'envie de découvrir de plus près comment tout cela fonctionne.

### 2. Le matériel

Le matériel fourni pour le projet était presque complet pour la session de cette année. Seul le terrain et les conditions d'éclairage n'étaient pas encore entièrement définis. Bien sûr des modifications peuvent encore être apportées au matériel existant. Contrairement aux années précédentes, l'intérêt est évidemment de tester en environnement réel le résultat du développement. La motivation des participants n'en est que plus forte.

De plus, par rapport au test sur simulateur, l'utilisation d'un environnement réel permet de prendre en compte toutes les contraintes physiques. Le poids du câble, l'irrégularité du terrain, sont autant de facteurs qui influencent les trajectoires et la vitesse de déplacement. Les ajustements nécessaires pour prendre en compte ces paramètres changent complètement la physionomie des algorithmes.

L'ensemble des équipements mis à disposition (ordinateur, robots, objets du jeu) sont bons et permettent de maintenir la concentration sur le développement.

### 3. La liberté

La philosophie du projet est de laisser le maximum de liberté aux 2 équipes. Il en résulte une grande flexibilité dans l'organisation et dans les technologies utilisées. Cela demande également plus de rigueur pour s'imposer un suivi régulier de la progression. Egalement, cela implique un risque plus important de faire le mauvais choix à un instant donné du projet (choix d'une technologie, choix d'un type d'algorithme, …). Cependant cela place les étudiants en position réelle de gestion de projet. Il faut apprendre à se relever suite à une erreur. La seule contrainte étant la satisfaction du client final.

### 4. L'entraide

La forme du projet avec opposition de deux équipes produit un effet intéressant : l'entraide. Tout au long du projet nous avons discuté avec les membres de l'équipe "adverse" des problèmes rencontrés. Certains aspects étaient mieux compris d'un côté et inversement. Contrairement à l'esprit de compétition, cela a conduit à une véritable coopération produisant des résultats meilleurs des deux côtés.

# B. Les problèmes

Les projets spécifiques d'une manière générale permettent aux étudiants de consacrer une part importante de leur temps à l'élaboration et à la réalisation d'un projet qui leur tient à cœur et qui les motive tout particulièrement. En ce sens, le projet SCOR représente une alternative intéressante face aux projets de conception classiques proposés en 5IF. Les aspects technologiques abordés dans ce projet sont réellement enrichissants et ne sont pas ou peu abordés dans le cursus IF classique.

## 1. Temps

Le projet SCOR englobe une quantité de notions abstraites pour l'étudiant IF très importantes. Analyse d'image (filtrage, espaces colorimétriques, tracking d'objets, reconnaissance d'objets, algorithmes de transformations, …) et intelligence artificielle sont les deux principales. Ces notions nécessitent un investissement colossal en temps pour être appréhendées, et appliquées à un projet informatique de ce type. La documentation est disponible en abondance sur Internet, sur des sites spécialisés le plus souvent en Anglais. Le début du développement est intervenu très tard dans le cycle de vie de ce projet de part cette autoformation nécessaire. La quantité de travail que représentent les projets 5IF n'a laissé que peu de place à un investissement de ce type.

## 2. Matériel

### Robots

Le choix des robots SCOR est justifié par plusieurs facteurs et notamment pour la simplicité de leur API et de leur prise en main générale. Néanmoins ces derniers présentent un certain nombre de défauts qui ont été très difficiles à corriger lors de ce projet.

Les robots SCOR sont très légers. Ils ne sont ainsi pas capables de pousser un objet relativement lourd. Aucun effet de frappe n'est possible sur la balle utilisée. Ainsi, le jeu de foot qui représentait l'objectif initial n'est pas envisageable avec ce type de robots. Il est uniquement possible de pousser la balle vers une objectif mais aucunement de l'envoyer vers un autre robot ou de viser une cage de but.

La légèreté des robots posent également des problèmes vis-à-vis du terrain. Les roues ne semblent pas tourner à la même vitesse malgré des commandes identiques envoyées aux deux moteurs. Les robots sont ainsi trop sensibles aux défauts du terrain et au poids des câbles qui les raccordent à l'ordinateur.

### Terrain

Comme nous venons de le voir, le terrain était construit de manière imparfaite. Le terrain n'est, par exemple, pas totalement plat. Les variations par rapport à l'axe horizontal peuvent paraitre négligeables mais, étant donné le poids très faibles des robots, elles se font sentir assez vite. Les robots tournent inévitablement du même côté avec des vitesses des deux roues identiques.

### Webcam

La Webcam en elle-même semble suffisante. Néanmoins, nous n'avons pas réussi à la faire fonctionner dans les hautes résolutions indiquées sur l'emballage et la notice. L'analyse de l'image est ainsi plus difficile et moins efficace. Sans en être certains, nous pensons que ce problème est dû à un pilote inadapté sous l'environnement Linux.

**Eclairage**

L'éclairage a régulièrement posé des problèmes quant à l'analyse de l'image. Les filtres que nous avons appliqués s'avéraient parfaitement adaptés à certains moments de la journée et complètement inefficaces à d'autres. Nous étions en effet très dépendants de la lumière extérieure en fonction de la météo et de l'heure. Certains reflets cachent presque les objets sur le terrain en fonction du soleil qui rentrent dans la salle par les fenêtres.

## 3. Pour que cela aille mieux…

La plupart de ces problèmes peuvent être corrigés.

En ce qui concerne le temps, il faudrait envisager d'éventuellement travailler à 6 ou 8 personnes dans une même équipe sur ce projet. Regrouper les deux équipes qui étaient cette année en concurrence pourrait permettre une meilleure répartition des tâches, plus d'entraide et plus de motivation pour chacun.

En ce qui concerne les problèmes matériels, les robots peuvent difficilement être changés. Peut-être faudrait-il dans ce cas changer les règles du jeu. Le terrain mériterait quelques bricolages pour le perfectionner. Le rééquilibrer notamment. Enfin, en ce qui concerne l'éclairage, l'installation de spots lumineux de forte puissance permettrait de normaliser la lumière quelle que soit l'heure du jour ou de la nuit.